Software Architecture

Lecture 4
Data Flow Systems

Rob Pettit
George Mason University
Previously: software engineering

Software architecture
- Packages -> Product Lines
- Software development environments
- Inheritance
- Abstract data types
- Programming-in-the-large
- Information hiding
- NATO SE conference

Programming-in-the-small
- 1950
- Subroutines
- Separate compilation
- Programming-any-which-way

Programming-in-the-large
- 1960
- Abstract data types
- Information hiding
- NATO SE conference
- Programming-in-the-large

Programming-in-the-WWW
- 1970
- Objects
- Inheritance
- Abstract data types
- Programming-in-the-large
- Information hiding
- NATO SE conference

Programming-in-the-physical-world
- 1980
- Heavily distributed systems
- Integrated product lines
- Component-based systems
- Software architecture
- Packages -> Product Lines
- Software development environments
- Inheritance
- Abstract data types
- Programming-in-the-large
- Information hiding
- NATO SE conference

Programming-in-the-WWW
- 1990
- Self-aware and adaptive systems
- Service-oriented systems
- Heavily distributed systems
- Integrated product lines
- Component-based systems
- Software architecture
- Packages -> Product Lines
- Software development environments
- Inheritance
- Abstract data types
- Programming-in-the-large
- Information hiding
- NATO SE conference

Programming-in-the-physical-world
- 2000
- 2010
- Computing embedded on everyday objects
- Computation in mobile phones
- Self-aware and adaptive systems
- Service-oriented systems
- Heavily distributed systems
- Integrated product lines
- Component-based systems
- Software architecture
- Packages -> Product Lines
- Software development environments
- Inheritance
- Abstract data types
- Programming-in-the-large
- Information hiding
- NATO SE conference

adapted from Mary Shaw
Previously:
Software Architecture

- representing the **structure** of a system from different angles: **views**

- **views** help manage the complexity of describing the architecture

- **viewtypes**
  - determine the kinds of things a view talks about
  - three primary viewtypes: module, C&C, allocation

- some **styles** occur frequently within each viewtype
  - module: decomposition, generalization, layered, ...
  - C&C: pipe & filter, client-server, pub-sub...
  - allocation: deployment, work assignment...
many styles
in the C&C viewtype

data flow
batch sequential
dataflow network (pipe & filter)
acyclic, fan-out, pipeline, Unix
closed loop control
call-and-return
main program/subroutines
information hiding - objects
stateless client-server
SOA
interacting processes
communicating processes
event systems
implicit invocation
publish-subscribe
data-oriented repository
transactional databases
stateful client-server
blackboard
modern compiler
data-sharing
compound documents
hypertext
Fortran COMMON
LW processes
hierarchical
tiers
interpreter
N-tiered client-server
outline

- data flow styles
  - process control
  - batch-sequential
  - pipe & filter

- lab 1

- pipe & filter sub-styles & implementation
data flow styles
assume:

- **connectors** are data streams
  - interfaces are reader and writer roles
  - transport data from writer roles to reader roles
  - unidirectional (usually asynchronous, buffered)

- **components** do not know the identity of upstream/downstream producers/consumers
  - read data from input ports
  - compute
  - write data to output ports
in data flow styles
availability of data controls the computation

- any component that has input may process it
- overall data transformation is the “functional composition” of individual transformations

\[ h(g(f(s))) \]
data flow styles
structure is an arbitrary graph

in general, data can flow in arbitrary patterns

however, frequently data flows in “stages”

or in simple, highly constrained cyclic structures
three major data flow styles

- **process control**
  - looping structure to control environment variables

- **batch sequential**
  - sequential processing steps, run to completion
  - typical of early MIS applications

- **pipe & filter**
  - incremental transformation of streams
example of process control
open-loop control system
example of process control

closed-loop control system

Ambient Air in

Heated Air out

Thermostat

Chimney

Gas Line
process control notions

- open-loop system:
  process variables not used to control the system

- closed-loop system:
  process variables used to control the system
  - controlled variable: goal
    (ex: air temperature inside the house)
  - set Point: value for the controlled variable
  - input variable: what the system can measure
    (ex: temperature of the outside air coming into the furnace)
  - manipulated variables: what the system can affect
    (ex: turning the furnace on or off)

- feedback control
  controlled variable is measured and taken into account
architecture of closed-loop process control

set point value \rightarrow control task \rightarrow manipulated values \rightarrow Process to Control \rightarrow controlled value

inputs ... feedback

legend
- computation (ellipse)
- data flow (arrow)

SWE 443 - Software Architecture
Data Flow Systems - 13
example architecture

heating closed-loop control

outside temp

desired temp

control task

on/off

furnace

temp inside the house

outside temp

desired temp

control task

on/off

furnace

temp inside the house

legend

computation

data flow
example architecture
simplified avionics

control task

pilot inputs:
Roll  Pitch  Yaw

trim

altitude

air speed

ailerons ctl. process

elevator ctl. process

rudder ctl. process

A to D conv

servo value

legend

+/-  digital positive or negative increment

~  analog signal
three major data flow styles

- **process control**
  - looping structure to control environment variables

- **batch sequential**
  - sequential processing steps, run to completion
  - typical of early MIS applications

- **pipe & filter**
  - incremental transformation of streams
batch sequential assumes:

- connectors
  - data is transmitted as a whole between steps

- components
  - processing steps are independent programs
  - each step runs to completion before the next step starts
examples of batch sequential systems

- classical data processing
  - payroll computations
  - IRS tax return computations

- early code compilers
three major data flow styles

- process control
  - looping structure to control environment variables

- batch sequential
  - sequential processing steps, run to completion
  - typical of early MIS applications

- pipe & filter
  - incremental transformation of streams
pipe & filter
assumes:

- connectors, called pipes:
  - move data from a filter output to a filter input
  - one-way, order-preserving, data-preserving
  - system action is mediated by data delivery

- components, called filters:
  - incrementally transform the input data to output data
    - enrich data by computation and adding information
    - refine by distilling data or removing irrelevant data
    - transform data by changing its representation
  - operate independently/concurrently among each other
    - no external context in processing streams
    - no state preservation between instantiations
    - no knowledge of upstream/downstream filters
    - topology determines the overall computation, not relative speed/CPU allocation of filters
example of pipe & filter system
telemetry data collection

Telemetry System

Sensors
(a few or a few thousand)

PRESSURE 1
PRESSURE 2
TEMP 1
TEMP 2

Airborne System

Recorders
Radio, Cable, etc.

Ground System

Time

Computer Displays

major frame decommutation

measurement units

apply coefficients

frame collection
time tag frame

record data
display data

minor frame decommutation
example P&F
autonomic vehicle
special case of the pipe & filter style

Unix pipes

- pipes: buffered streams supported by OS
  - assume ASCII character streams
  - can treat files as well as filters as data sources and sinks
    - the good news: anything can connect to anything
    - the bad news: everything must be encoded in ASCII, if not, it must be “translated” before piping

- filters: Unix processes
  - built-in ports: stdin, stdout, stderr
  - filters by default transform stdin to stdout
example of Unix pipes system

```
cat /etc/passwd | grep "joe" | sort > junk
```
special case of the pipe & filter style

Yahoo pipes

- web application for authoring *Yahoo pipes*
- *Yahoo pipe* is a data mashup
  - application hosted by Yahoo
  - combine, filter, and present data from different web sources
outline

• data flow styles
  • process control
  • batch-sequential
  • pipe & filter

• lab 1

• pipe & filter sub-styles & implementation
outline

- data flow styles
  - process control
  - batch-sequential
  - pipe & filter
    - case study: Tektronix

lab 1

- recover views from the code: module, C&C

- pipe & filter sub-styles & implementation
lab 1: pipe & filter system
build avionics instrumentation systems

- data comes in from airplane sensors

<table>
<thead>
<tr>
<th>ID</th>
<th>Data Descriptions and Units</th>
<th>Type</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Time: number of milliseconds since the Epoch (00:00:00 GMT on January 1, 1970)</td>
<td>long int</td>
<td>8</td>
</tr>
<tr>
<td>01</td>
<td>Velocity: airspeed of the vehicle, measured in knots per hour</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>02</td>
<td>Altitude: vehicle's distance from the average surface of oceans, measured in feet</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>03</td>
<td>Pressure: atmospheric pressure external to the vehicle, measured in PSI</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>04</td>
<td>Temperature: temperature of the vehicle's hull, measured in degrees Fahrenheit</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>05</td>
<td>Pitch: angle of the nose of the vehicle, if positive, the vehicle is climbing</td>
<td>double</td>
<td>8</td>
</tr>
</tbody>
</table>

framed as

```
00000 Time 0001 Velocity … n data
00000 Time 0001 Velocity … n data
...```

![Image of data framed as a sequence of bytes]
lab 1: pipe & filter system
build avionics instrumentation systems

- **system A**: reads flight data and
  - converts Temp to Celsius
  - converts altitude to meters
  - removes other fields

- **system B**: same plus
  - include all fields
  - removes wild altitude variations >100m
    and replaces them by interpolated values (avg of previous and next)

- **system C**: merges streams from two sets of sensors
  - output frames are sorted by time
lab 1: pipe & filter system
based on a code framework

remember:
- module and C&C view types show different aspects

module view

C&C view

Legend

Pipe

Filter

Module

is a

uses
lab 1
module view

Object!

Thread

Plumber

OutputStream

InputStream

FilterFramework

SourceFilter

MiddleFilter

SinkFilter

PipedOutputStream

PipedInputStream

PipedOutputStream

PipedInputStream

app class

extends (is a)

uses

Java lib

app class

extends (is a)

uses

SWE 443 - Software Architecture
lab 1
C&C view

- Filter1 (<SourceFilter>)
- Filter2 (<MiddleFilter>)
- Filter3 (<SinkFilter>)

Plumber

- creates and connects the filters
- doesn't intervene during system operation
  - therefore not normally represented

main
filter (thread)
pipe
outline

- data flow styles
  - process control
  - batch-sequential
  - pipe & filter
    - case study: Tektronix

- lab 1

- pipe & filter sub-styles
  & implementation
in data flow styles
data flow dominates structure and control

- a data flow system is one in which:
  - the structure of the design is determined by the motion of data from component to component
  - the availability of data controls the computation
  - the pattern of data flow is explicit
  - this is the only form of communication between components

- variations on this theme
  - how control is exerted (e.g., push versus pull)
  - degree of concurrency between processes
  - granularity of computation
  - topological restrictions (e.g., pipeline)
pipe & filter sub-styles & implementation

system level

- topological constraints
  - some styles insist on a pipeline, or no cycles

- strategies for creation of elements
  - centralized versus distributed
  - static versus dynamic
  - declarative versus operational
    (some systems have both)
pipe & filter sub-styles & implementation components

- concurrency
  - separate processes or single address space

- constraints on ports
  - e.g., Unix components typically have 3 ports: stdin, stdout, stderr
  - e.g., some styles require single input and single output port

- computational model
  - e.g., how do you deal with multiple inputs?
pipe & filter sub-styles & implementation
connectors

- finite versus infinite buffering
- dealing with end of data
  - can a writer terminate the flow of data?
  - can a reader choose not to consume data on a pipe?
  - what should a reader or a writer do after data has been terminated?
- does data have different types?
- how many kinds of pipes are there? (colored pipes)
references for implementation strategies

- **book on Java threads**
  - *Concurrent Programming in Java: Design Principles and Patterns* (2nd Edition) by Doug Lea

- **online tutorial on Java threads**

- **Design Patterns**
  - book by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
in summary
select a data flow style when:

- task is dominated by the availability of data
- data can be moved predictably from process to process

pipe-and-filter architectures are good choices for many data flow applications because

- they permit reuse and reconfiguration of filters
- generally easy to reason about
- reduce system testing
- may allow incremental AND parallel processing

there may be a performance penalty when implementing data flow styles over a single process
in summary
styles are rarely usable in simple pure form

- one technique is to specialize styles
  - styles become more constrained, domain-specific
  - trade generality (expressiveness) for power (analytic capability)